Combat System Architecture

Mr. James D. Horner

Combat system architecture is defined in terms of conveying requirements utilizing specific modeling elements generated during development. Combat system requirements are characterized as conveying either performance- or quality-type attributes. Architecture constructs are shown as potentially consisting of functional, design, or implementation elements. Functional elements represent a refinement of top-level performance requirements. (Conceptual) design elements define the system interfaces needed for conveying system quality attributes. Architecture may also include implementation elements of those system functions that are of high concern. A combat system functional model is presented that identifies the primary decision and control processes. A consistent knowledge base is identified in this model as key in automating supervisory functions. A combat system design model is also presented. This model derives from the functional model and identifies the major combat system (functional) interfaces. A discussion is provided of each of the major combat system functions.

Introduction

A properly defined architecture is key in developing cost effective and capable combat systems. It establishes the technical vision of what the combat system needs to do and how it is to do it. It clearly conveys to the designers the important requirements and their performance measures. Most importantly, architecture must capture the organizational and design concepts so they can be modified and enhanced in the future.

In this article, architecture refers to combat system hardware (HW), software (SW), or devices. That is, it is assumed that some initial functional allocation has been made between system HW/SW and ship personnel. For ship systems, these allocations need to be supported through process analysis of shipboard operations. Through operational analysis, ship functions and their timing attributes are identified and defined. Allocations are made of those functions to either humans or machines based on life-cycle cost versus performance trade-offs.

Defining architecture means determining those aspects of the combat system that are to be specified by the architecture. Fundamentally, it is the system requirements that must drive the architecture design. Thus, what needs be identified are those combat system requirements that should be specified at the architectural "level." It also needs to be determined how these specifications may be best represented.

A basic criterion for architecture is that it is useful for designing and/or implementing the combat system. Thus architecture is generally thought of as a top-level system design. However, depending on the system and its requirements, there may be other useful elements. As an example, it may be appropriate in some instances to mandate the use of certain low-level interface standards prior to development. This may be the case for systems with relatively lax performance requirements but very stringent interoperability requirements. However, for other systems, low-level interface requirements would more appropriately be defined during detailed design or implementation.

Because architecture definition is part of development, examination of the development phases should reveal likely candidates for combat system architectural elements. The system development phases or activities may be broken down into requirements analysis, functional analysis, design, and implementation. Thus, an architecture may specify either design or implementation aspects of the combat system. In addition, the architecture may contain elements from the functional analysis phase.

Thus, it is important to identify those elements from the various development activities that may compose the combat system architecture. It is also important to identify the types of combat system requirements that are appropriately conveyed by those elements. It will be shown that the elements required for a combat system architecture are very much dependent on the nature of the combat system and its critical requirements.

COMBAT SYSTEM ARCHITECTURE REQUIREMENTS

Architecture addresses some set of key requirements of the combat system. That is, the combat system architecture specifies certain system design or implementation characteristics or constraints. A general characterization of combat system requirements may thus be useful in identifying architecture elements. From this characterization, those requirements that may best be specified at the architectural level can then be identified.

Combat system requirements can come from many different sources and considerations. However, many types of requirements are implied through operational analysis. That is, mission requirements infer operational requirements that in turn infer requirements onto the combat system. In particular, much of the system ***performance***-related requirements discussed below are inferred through operational analysis. In this respect, it is important that operational analysis is performed in such a way as to maintain traceability throughout the phases of a requirement's refinement.

Performance requirements are defined here as those that address behavioral, interface, or environmental aspects of a combat system. Interface may refer to both the combat system external and internal interface requirements. External interfaces include external system communications protocols, legacy system interfaces, and human-computer interfaces (HCIs). Requirements for interoperability establish many of the external-interface-type requirements. External interface requirements could also include a host of other system constraints that allow for such things as hookups for power or standards for physical mounting and transportation.

Behavioral requirements refer to requirements for how the combat system must interact within its defined environment. Behavioral requirements specify how and when the system must respond. This includes functional and timing requirements. At the (combat system) context level, behavioral requirements are identified through analysis of scenario definitions and threat descriptions. Finally, performance requirements include the environmental requirements of the combat system. These requirements specify the environmental conditions within which the combat system must operate (effectively). Environmental conditions include temperature, humidity, shock, vibration, moisture, etc.

There are other types of combat system requirements that, according to the definition provided, are not strictly performance requirements. One class of these is the requirements for availability and reliability (ARM). Requirements for availability are typically divided into inherent availability (Ai) and operational availability (Ao). Ai generally refers to the percent time the system would be operational without maintenance. Ao refers to the percent time the system is required to be operational, given that certain specified maintenance procedures are followed.

Requirements for reliability address the need for the system to be operational during critical times. That is, regardless of the system's Ao, it must also maintain some minimum capability during mission-critical times. Requirements for fault tolerance address many system attributes that can impact both reliability and availability. Fault tolerance is an important quality attribute of mission-critical systems.

There is another important class of quality-related requirements that address combat system life-cycle costs. These are requirements for system flexibility, upgradability, and maintainability. System requirements for maintainability address (owner) costs of maintenance. Maintainability is often grouped with ARM, accentuating the trade-offs involved between quality and cost.

Flexibility generally refers to the ease with which the combat system may be reconfigured to support alternative (future) shipboard operations. Upgradability is the ease to which future enhancements or technology updates may be accomplished. These types of requirements are difficult to directly quantify because they involve predictions of technology trends. However, there are (architectural) characteristics of a combat system that can be used to generate relative measures of flexibility and upgradability. Examples of these characteristics are the degree of modularity and interface standards use.

Thus, three major categories of combat system requirements have been identified. Performance requirements identify the basic behaviors and characteristics required of the combat system. The second category identifies quality attributes related to fault tolerance, reliability, and availability. The third category identifies those quality attributes related to life-cycle ownership costs. Part of the requirement's definition process is defining the appropriate degree of requirement's specificity. Similar to the requirement's definition process, this is a continual refinement process that requires knowledge of applicable technologies (and the requirements themselves).

ARCHITECTURE FUNCTIONAL ELEMENTS

Functional modeling is an important phase of the development process. This is true regardless whether the models are considered necessary in establishing (architectural) requirements or some element of the architecture itself. Functional models provide a structured means of analyzing and conveying many and much of the performance-type requirements of a system.

Functional analysis is not design. Design is a synthesis process and in this regard, is opposite to functional analysis. The structured analysis models and specifications resulting from functional analysis are implementation independent. That is, they do not imply any particular physical realization. Although functional analysis is implementation independent and is not design, it cannot be considered design independent. Indeed, functional analysis is a primary means of establishing design requirements.

Functional analysis is a continuation of requirements analysis and is required for understanding system functional requirements, their related performance factors, and for generating verifiable specifications. At the requirements analysis phase, the combat system is initially viewed as a "black box." Requirements produced at this level of abstraction may or may not be verifiable. In addition, an understanding of internal system functions is typically required to gain confidence of black box specifications. This depth of understanding can be achieved only through functional analysis.

It is through functional analysis that the core or essential characteristics of combat systems can be modeled prior to design. Today, functional analysis for complex systems is primarily accomplished through structured analysis methods. The system descriptions resulting from these analysis methods are often referred to as *essential models* because they capture the essential nature of the system. The models provide well-defined conceptualizations of a system's data/information, functions/processes, and timing/control aspects.

Initially, functional analysis is focused on (what turns out to be) the application SW of the system. That is, the analysis must first reveal what the system must do. The application-level functional models can then be used to infer requirements for a supporting "middleware," operating system (OS) and/or HW capability. Middleware includes communications, data management and other "common services" and utilities to support system applications.

Functional modeling is typically done at levels below the applications layer in those cases where these layers may need to be custom built. That is, when the requirements of the system are such that a commercial off-the-shelf (COTS) or other reuse-type component may not be adequate. As an example, functional analysis is rarely performed at all when instituting a networked personal computer (PC) system aimed at business use. Alternatively, the primary application SW in certain warfare areas may itself be middleware. That is, in cases where the primary requirement is to provide communications or data-management capabilities, it is important to focus the analysis on these types of functions.

Thus, there can be inferred a general guideline for inclusion of functional modeling elements resulting from functional analysis in a combat system architecture. The guideline is to include those functional modeling elements that establish the key system requirements (design drivers). As an example, command, control, communications, computers, and intelligence (C4I) systems can be viewed as data driven. That is, the core requirements are those for storing, distributing, and otherwise managing data. Thus, the data (or object) schema of the C4I system should certainly be included as part of the architecture. In addition, the C4I system architecture should include general descriptions of the mechanisms required for controlling data updates and access.

Alternately, an antisubmarine warfare (ASW) system design may be characterized as functionally driven. Although ASW systems certainly have requirements for real-time and data management, it is the fusion algorithms and tactical decision aids that drive system design. Because of the nature of the data, sophisticated algorithms are required for sensing and tracking contacts. In addition, complex applications are needed for analyzing the underwater environment and providing tactical decision support. Thus, in the case of ASW, the functional analysis should concentrate on identifying, defining, and classifying functions. These functional descriptions would be a key element to the combat system architecture.

Another combat system may have stringent timing and control requirements. For example, in antiaircraft warfare (AAW), there are stringent timing requirements for both the processing of measurement data and countering threats. However, because AAW data becomes quickly outdated, there are minimal requirements for data management in the traditional sense. Indeed, because of the character of the data, AAW data management distills down into a performance-driven data distribution problem. Thus in the case of AAW, the analysis results of the timing and control aspects of the system should be represented by the combat system architecture.

It is noted, however, that stringent timing requirements tend to impose stringent requirements on all aspects of system design. Detailed functional analysis is often required at all architectural levels of the system for appropriate specifications. That is, it cannot be assumed that there exists an adequate COTS solution or interface standard for any system components, including HW. Those COTS components that are incorporated may need to be utilized or integrated in unusual ways. In addition, sophisticated process control mechanisms, such as prioritization and synchronization, are often required.

Thus, the amount and type of functional modeling that is used to establish combat system architecture is dependent on the nature of the particular requirements. In addition, many of today's ship systems are being reengineered or enhanced for new capabilities or mission areas. For most of these legacy systems, there may have been no formal functional analysis completed. In cases where functional analysis was performed, the methods used might be considered by today's standards as outdated, unstructured, or not applicable to current technologies.

In any case, functional analysis of legacy systems is necessary to gain the level of understanding required for either enhancing or reengineering. It provides a means of viewing current capabilities at various levels of detail and insight into the coupling between system components or (SW) modules. In addition, functional analysis can identify the undocumented performance requirements of a system. Thus, in the case of legacy system reengineering, the results of functional analysis would be a key element to the architecture description.

Functional analysis produces models that are capable of capturing and conveying performance-type requirements for system behavior and interfaces. However, functional analysis is generally performed only for new developmental items. That is, it is performed only for those combat system functions and to the level of detail needed to establish verifiable system specifications. The type of system functions analyzed and the amount of detail required vary according to the specific nature of the combat system requirements.

ARCHITECTURE FUNCTIONAL MODEL

Functional analysis provides methods for analyzing the functional, data, and control nature of a system in a top-down fashion. However, a combat system functional concept is extremely useful in establishing a reasonable starting point for the analysis. A combat system concept can convey the lessons learned in past analysis and design efforts. It provides a high-level partitioning of system functions that provides guidance for performing the analysis. Without such guidance, many iterations of functional decomposition and synthesis are required to establish a reasonable top-level system partitioning.

A starting point for modeling the functional nature of a combat system is the H-Architecture. The H-Architecture identifies sense, control, and engage as the three primary functions of a combat system. This is important in that it identifies warfare as primarily a control problem. The H-Architecture is also fractal in nature. That is, combat systems have core control elements at all levels of command and operation. Indeed, even in those cases of "smart weapons," the Sense, Control, and Engage Model is valid.

In today's Navy, there is an emphasis on crew reduction and human-centered design. Both of these can be viewed as thrusts to develop more cost-effective systems. The emphasis is in obtaining greater system efficiencies through automation and by incorporating the operator as part of the system design.

Architecture has been defined for the purposes of this article as referring to those combat system functions not allocated to the human operators. However, achieving either of the above objectives for efficiencies and effectiveness requires consideration of the human functions in combat systems. That is, the combat system functional concept should not bias resource allocation decisions towards either a human or HW/SW solution. In this way, resource allocation decisions can be revisited throughout development.

Figure 1 presents a conceptual functional model of the control part of a combat system that attempts implementation neutrality. A good way to view the model is that of a generalized controller for complex systems. In this control model, knowledge supports decision-making (through information processing) that, in turn, supports control.

The combat control model of Figure 1 is conceptual in two ways. First, the functions generally apply only to either human functions or application-level control SW. That is, it does not include inferred system functions. Second, it does not have the formalism of a structured functional model. In this regard, it represents a combat system functional concept for the purpose of providing guidance for functional modeling. Note that similar to design modeling, functional modeling can iterate from the conceptual to the detailed level.

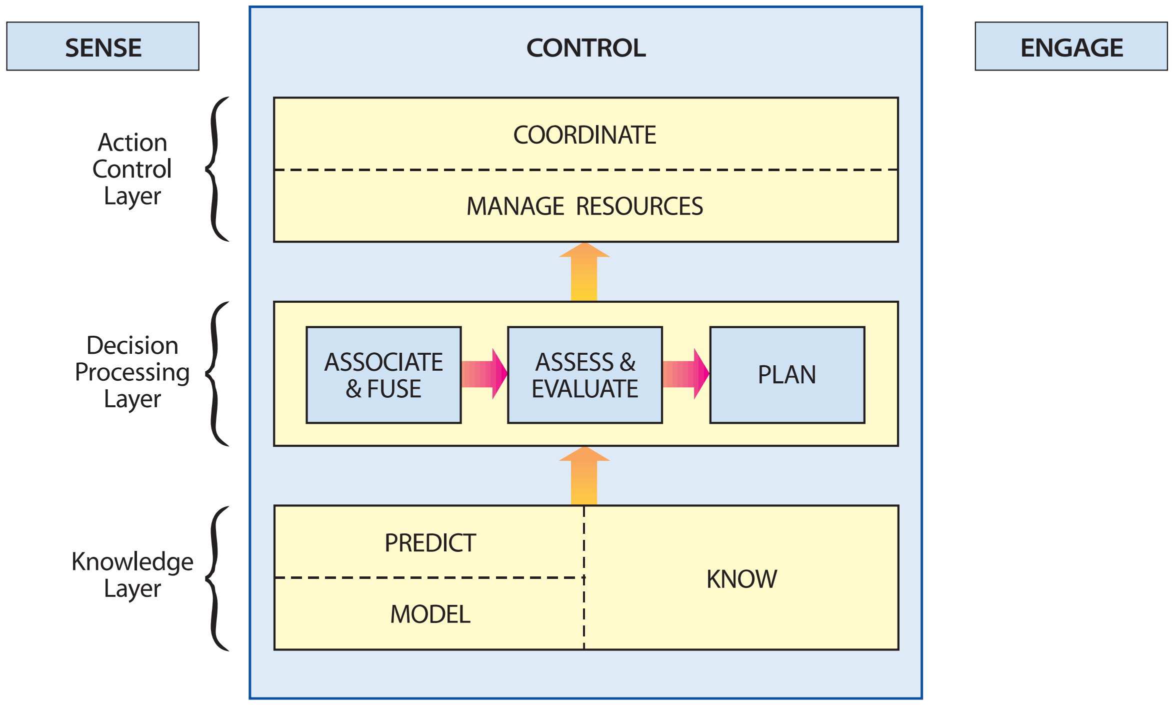

Figure 1—Combat Control Functional Model Concept

A functional model should capture two key aspects of combat system architecture. First, it should capture the process nature of the system (information processing and control). Figure 1 identifies two primary combat system process flows. Tactical information is processed (left-to-right flow) in the *Decision Processing Layer*. In addition, this figure models a control support process (down-to-up) whereby the lower (functional) layers provide support to the next higher layer. These are two distinct, but interrelated and coupled, processes.

The second key aspect to architecture is structure. There are structural or organizational aspects to combat system architecture at all levels, from the command structure to the computing system structure. However, architecture structure can be seen as an alternate view of process. As shown in Figure 1, process flow can be arranged in such a way that conveys the structural or organizational aspects of the system. For example, a process model of the *Action Control Layer* can be constructed in such a way that it conveys organizational requirements of the command structure.

Process modeling, then, can be used for defining both process and structural views of combat system architecture. However, this requires an expanded definition of the control aspects of combat systems. The *Action Control Layer* of Figure 1 controls the internal combat system processes in addition to the combat resources (sense and engage). In particular, it controls the tactical situation through effective control of combat system processes.

Modeling the internal control (and decision-making) processes as they relate to control of warfighting assets can establish both the process and structural aspects of combat system architecture. However, it is both the functional nature of the control-decision support systems and the capabilities of the assets available that will determine the control structure of the ship. Thus, it is necessary that the combat system design be performed concurrently with that of the ship command and control system.

Control & Coordination Layer

The *Action Control Layer* of Figure 1 models the high-level control aspects of a combat system. Assuming that the fundamental purpose of a combat system is to maintain (tactical) control, then the *Action Control Layer* must establish the core requirements for the remainder of the system. That is, the requirements for warfare control are the key combat system requirements.

The *Action Control Layer* provides for ship and ship resources control. This includes control over **Sense and Engage** functions. Control or management of device-type resources is a relatively low-level monitoring, scheduling, and control function. Alternately, management of personnel requires inputs such as status reports and recommendations. Personnel are managed through orders or other types of guidance. Thus, resource management can have both supervisory and nonsupervisory aspects. These two different aspects of control need to be identified and properly characterized in order to consider it, or portions of it, for automation.

The *Action Control Layer* identifies a second control-type function or activity termed **Coordinate**. This function acknowledges that intelligent control over resources inevitably involves coordination with other intelligent control agents. Thus, combat system functions that coordinate between control agents are modeled as distinct from those directly involved in the control of ship resources. Effective design of coordination functions requires consideration (analysis) at all levels of command and operations. Determining all possible external impacts of coordinating the control of resources is, in itself, a difficult control problem. However, it is important because control coordination establishes the context for the combat system control problem.

Another difficult aspect to control is in determining the most effective time to react. In cases where there is a need for quick, reactionary control, decisions can be justifiably made without the benefit of planning. Thus, combat control must be based on some adaptable weighting of the current scene (and its history), assessments that have been made, and current combat plans (see *Decision Processing Layer* of Figure 1). Humans are particularly good at performing adapted or situational-based weighting. However, as ship control systems become more automated, greater analysis will be required of this difficult aspect to complex control.

Functional and process analysis can be used to model the command and control necessary to achieve the various mission objectives. That is, the analysis can identify effective paths of control through the command structure to the warfighting assets. The analysis can also identify the information requirements (from the *Decision Processing Layer*) needed to support command and control decisions. These kinds of control analysis are required for new combat systems development. However, they are also critical for achieving integrated capability across multiple warfare areas. These command and operational control models are thus important elements of combat system architecture.

Decision Processing Layer

The *Decision Processing Layer* of Figure 1 produces the information required for decision-making. An important aspect of this layer is that each of the outputs of three major *Decision Processing* sub-functions should be available for real-time control (automated or otherwise). These functions provide to command the information and recommendations necessary for effective and timely control. Alternately, these functions must operate under command directive. Directives (such as guidance, orders, or acceptance of recommendations provided) control resources by controlling the flow and processing of information.

The **Associate & Fuse** functions associate and fuse measurement, track, and surveillance data. This requires the modeling of threat characteristics, ownship sensor performance, and the environment. In addition, the **Associate & Fuse** function implies an information and data management capability for the storing and distribution of tactical scene information.

There are important coordination aspects in maintaining a coherent and consistent tactical scene across warfare areas and command levels. Wider scene information from higher command levels should be provided for more optimal association of own data. The *Action Control Layer* coordinates this and the sending of partial scene information to the next higher level of command for association and/or fusion.

The **Associate & Fuse** functions must respond to control from the *Control & Coordination Layer*. One reason this is important is to control the level of fidelity of fused information being reported up the command chain. For example, it may in some instances be effective to provide relatively low-level sensor measurement data to shore-based command stations. However, doing so continuously would quickly overwhelm computing, communication, and operator resources.

Achieving the Navy goal of a shipwide coherent and consistent tactical scene will require analysis of both the data/information and control/coordination aspects to fusion. This is necessary to ensure that the appropriate data is fused at the appropriate (command) level and at the appropriate time. It is also necessary because automated association and fusion algorithms typically operate on the assumption of data independence. The control and data models resulting from these analyses are an important part of combat system architecture.

The **Assess & Evaluate** function provides assessments based on the current scene and its history. Threat assessments, situational assessments, kill assessments, etc., are performed within this functional area. Assessments derive information based on time-history patterns of behaviors and relations. Threat identifications and their intent can often be inferred from the scene and its history. In addition, certain threat formations can indicate likely future behaviors. Perhaps most importantly are evaluations of relative (ship-to-threat) strengths and weaknesses. This is important in indicating the appropriate posturing of ownship (defend, attack/ prosecute, monitor, avoid) and thus the type of plans that need generated.

At the tactical warfare level, the **Plan** function of Figure 1 conducts planning within the overall context of mission objectives. Tactical plans may need to be continuously modified based on current tactical situation. Plans reflect some weighting of ship postures driven by mission objectives and current situation. Assessment and planning functions have traditionally been allocated to humans. They most often require human-type knowledge and experience. The exception is in the AAW warfare area, where many assessment functions have been automated because of the stringent timing requirements.

Functional analysis can be used to identify common functionality across a system. Identification (and separation) of common functionality in the *Decision Processing Layer* is especially important. This is because common functions throughout a decision-making process must be made identical in order to ensure consistency. In fact, functional analysis of combat systems reveals that common functionality does exist and can be characterized as knowledge.

Knowledge Layer

The *Knowledge Layer* of Figure 1 recognizes the important role of knowledge in support of decision-making. In both humans and combat systems, knowledge and its role in support of decision-making are often assumed. Identifying knowledge functions is important in determining the ease in which combat system functions can be automated. It also reveals the need for knowledge consistency across system components—machine and human alike.

Knowledge is defined in this model as that which enables the prediction of future occurrences. In this sense, having knowledge about something implies that some aspect of its behavior can be predicted. It is this ability to predict that enables interaction with the environment in other than a reactionary mode.

Knowledge can pertain to knowledge of the physical environment, personnel, or man-made objects such as ownship or threats.

One type of knowledge is the ability to predict based on real-world models. The **Predict** function of Figure 1 provides the capability to project the tactical scene into the future. Note that this encompasses capabilities for predicting the performance of ownship assets, including personnel. The (physical) environment, objects, events and, most importantly, their uncertainties are projected into the future based on behavioral models.

It can be noted that each of the functions in the *Decision Processing Layer* depends on prediction capability. For example, measurement-to-track geographical correlation requires projection of a track to the time of current sensor update. Projection of track position and uncertainty is performed in accordance to appropriately selected platform motion models. Similarly, prediction capability is key to making situational assessments and conducting planning.

The second kind of knowledge in this control model is referred to as **Know**. This knowledge can be thought of as "rules of thumb" that have been gained from the past. For combat systems, they are rules that are represented heuristically in knowledge-based systems or deterministically in doctrine servers. It represents the capture of lessons learned through either real-world experience or off-line simulations. The **Know** functions can support the assessment and planning functions with doctrine-to-scenario matching capabilities. Although typically faster than real-time modeling and simulation, this type of knowledge can also be less flexible.

Segregation of knowledge functions is also important to combat system design. Knowledge functions require special treatment regarding maintenance. Knowledge functions also require special expertise in their development and are prime candidates for continual off-line upgrades. Knowledge, for both humans and machines, can become outdated and thus must be periodically checked for validity. Knowledge functions also have issues of ownership that are inherently different from many other types of SW.

Knowledge can take many forms in combat systems. As combat systems become more automated, issues regarding the use, development, maintenance, and validation of stored knowledge will become more acute. As regards architecture, knowledge required for effective performance of combat system functions needs to be identified. There are many combat system functions or algorithms that inherently have knowledge components. These include most "intelligent" or nongraphics-based tactical decision aids. For these functions, the knowledge components should be partitioned out. This is important for future systems maintenance and to ensure consistency across automated combat system functions.

ARCHITECTURE DESIGN ELEMENTS

Functional models include functional elements with specified relationships between the elements. Functional elements include the functions themselves, dataflows between the functions, and various types of control elements and other constructs to describe system timing. Functional modeling elements incorporate various attributes that define them and, in addition, have specific relationships to each other that together define the model. Like-elements typically are related by relationships that specify decomposition. For example, both functions and dataflows can be decomposed. Dataflows are related to functions in terms of input and output relationships. Triggers, a type of functional modeling element that effects the timing of functions, also are related to functions in terms of input and output. Additional control constructs are typically required to completely specify the timing aspects of a system in the functional model.

Combat system architecture may also directly address the important design aspects of a system. Indeed, architecture is often characterized as a top-level or conceptual-system-level design. System design first requires that remaining required system functions be derived from the functional models. In the case where functional modeling was done only at the application level, the amount of derived functionality may be extensive. In other cases, it may involve only HW or ancillary functions such as data recording and playback.

In design, the function, data, and control elements are organized into systems. System interfaces are identified and specified. At the lowest level, SW modules or HW components are designed in detail. Detailed design is often referred to as implementation. For SW modules this involves the writing of actual code for the functions. For HW components this involves designing the circuits or other types of components. At all levels of design, decisions are made regarding reuse.

Architecture design elements specify the interfaces between defined functions. This is different than architecture functional elements where system functionality, as opposed to system functions, are specified. In the case of architecture design elements, the functional analysis has been completed for the entire system. In addition, the interfaces between functions have been specified in terms of data formats and protocols. Alternately, in the case of architecture functional elements, the functionality of the system can be specified at any level of detail. That is, there may remain implied functionality that needs to be specifically derived. In addition, internal system interfaces may be identified but not necessarily defined in detail.

Many system requirements can be adequately addressed only during design. These include the requirements for ARM, flexibility, and upgradability. Note that each of these reflect “quality” attributes of the combat system. To the extent that a combat system architecture is characterized as a top-level design, it is primarily addressing the quality attributes of the system. Notions of maintainability, flexibility, and upgradability are difficult to quantify outside the context of system design. That is, without referring to a particular top-level or conceptual design, it is difficult to establish these types of system attributes or to estimate costs of their implementation.

However, requirements for maintainability, flexibility, and upgradability may be partially addressed through requirements for modular and open systems design. Modularity can have various forms. For SW, modularity is the key to well-written code. Modular code has the characteristics or attributes of data-independent functions, functionally partitioned, portable applications, common utility-support modules, and standardized interfaces. Object-oriented programming addresses many of these attributes. However, to meet quality-related requirements for application SW, it is most important that the customer also gain ownership of the source code.

Thus, many important requirements are specified in terms of top-level design. Similar to the functional models, it may be necessary to specify only basic functional organization at the application level (conceptual design). This is true for two reasons. First, application SW requirements can impose or infer requirements for quality on the system middleware, HW, and its OS. Second, these lower level support components are often standardized in terms of both functionality and interface.

However, it is not always possible to address total system requirements at the application SW design level. Doing so, very much depends on the particulars of the combat system. In the case of mission-critical systems with stringent timing requirements, a conceptual design of the entire system may be necessary to specify the quality attributes of the combat system.

Layered Services

One method commonly employed in organizing functions in a modular fashion is the client-server model. In this model, servers generally respond to requests from clients and clients wait on servers to respond to those requests. An important type of server is the data server that provides (well-defined) access control to the data. This type of architecture puts numerous constraints on the system design that can impact performance. The client-server organization may slow performance because the server can be a bottleneck for responding to data requests. However, it is considered a powerful means of simplifying system design and achieving, in the case of data servers, data-application independence (decoupling).

A layered architecture model can be viewed as a generalization of the client-server model. It is similar to the client server, but there are multiple layers of services, and the kind of service provided is not limited to data management. The basic constraint is that lower level services (functions) respond only to requests from higher level services and, preferably, only the next higher level. An example of this type of organization are the various communications protocol "stacks." Between each layer are well-defined and, whenever possible, standardized interfaces.

Each interface and service layer can potentially impart performance degradation on the system in terms of speed, capability, and flexibility. However, relaxing the requirement that services can be provided only by the next lower architectural level can greatly mitigate this; i.e., by allowing for "drill-down" through the services layers.

Object Design

Another powerful architecture design construct is objects. Objects provide an intuitive approach for combining functions (methods) and data (characteristics) into modules that can be better understood and thus can better support system upgrades and growth. The object paradigm also provides many (nonarchitectural) benefits for SW development. The most important of these benefits is reduced new-code development through various reuse mechanisms (inheritance, composition, and templates).

Another useful architecture feature of objects is that they allow designers to completely specify object interfaces before designing the object itself. The object interfaces are specified by defining the object class structure, which also establishes object inheritance relationships. Indeed, SW design using the object paradigm can be considered equivalent to defining the object classes. Writing the code for the object methods (functions) that maintain the characteristics (data) is then viewed as SW implementation.

There are three main areas of object design: application, data management, and communications object design. Each of these areas requires a different approach corresponding to the unique system capabilities they provide. In addition, application objects for HCIs and non-HCI SW can also have unique object design requirements.

Object-to-object communication supports development of peer-to-peer applications, whereby "client" modules can directly send requests to, and get results from, other modules. Similar to data servers, objects provide for standard mechanisms to access and, thus, protect their data. Objects' data is protected because their methods must be used to access their data. Thus, application objects can be architecturally viewed as specialized data servers.

Common-Object Request Broker Architecture (CORBA) provides a standard for high-level object-to-object communications. Similar to most high-level communications protocols, CORBA allows programmers to write communications interface code in source-level language. It also serves the purpose of making all objects in the system appear in the local address space, providing locality transparency. This is especially important when it comes to performing system upgrades, and in particular, porting application code.

Thus, object-to-object communications support message-sending between objects. This provides a means for objects to "trigger methods" in other objects. The data sent might be characterized primarily as control messages that can also incorporate the data to be processed. That is, the messages are generally requests for processing. The results of the data processing may be sent directly back to the requesting object or "put" into a database. This type of object-to-object communications is typical between two non-HCI-type applications.

However, there also exists actual object communications that are different than object-to-object communications. Object communications is the sending of complete objects, methods, and data. Objects are sent for execution within another process (or computing platform). This type of object communications is often used in distributed display systems. That is, the objects sent are typically display or multimedia-type objects. An example of object communications is the sending of JAVA Applets. Communicated objects typically are language-limited in the types of functions that they are capable of executing because of security reasons.

The basic functions of databases are data storage and data distribution. Databases also provide data access control. Databases provide a ready means for backup, recovery, and recording of systems' data. Unlike data communications, databases can establish and maintain relationships between the data.

Storage of the data (object characteristics) associated with application objects can be viewed as saving the state of the object. This is also referred to as providing for object persistence. One of the characteristics of objects is that they can be created and destroyed during code execution. In addition, object languages do not directly provide for object persistence. This makes data storage an especially important issue for combat system design where object programming is used.

The COTS object-oriented database management systems (DBMS's) available today may have the performance attributes required to support many combat system applications. Most provide standardized access through C++ extensions and have faster data access times than relational DBMS's. In addition, they can store complex data structures often associated with combat system data.

A key reason for developing architecture is to simplify the combat system design. One of the primary means of achieving design simplification is through modularization. For modern SW design, there are three basic approaches to modularity, each of which needs to be incorporated. The first approach is to partition the system into like functionality. The second approach is to partition the SW into service layers. The third approach is to use object-oriented design and programming.

Modularity also entails defining the interfaces between system modules or boundaries. In effect, the interfaces define the required functionality and performance of the modules. Each interface or interface type needs to be standardized. Interface standards can be either adopted or defined. However, it is important to note that specifying particular system functions is a high-level design activity. This is in contrast to defining the required system functionality, which can be better characterized as analysis.

ARCHITECTURE DESIGN MODEL

Figure 2 presents a conceptual design for the control SW of a ship combat system. This model indicates the kind of functional partitioning that is necessary for a conceptual design model. The conceptual design model should identify the major subsystem interfaces that need to be defined. It should be noted that the design of each functional area requires very different types of skills and knowledge, both technical and domain. Each arrow type in Figure 2 corresponds to an interface type to be defined according to the specifics of the system requirements. The specifying of major module interface standards may need to be part of the system architecture definition.

Applications Layer

The *Applications Layer* of Figure 2 is differentiated from the *Support Services Layer* in that *Applications Layer* functions or modules do not interface directly with the OS. Application modules are exclusively written in source-level code and interface with the *Support Services Layer* via standardized application programming interfaces (APIs). The use of object programming for application modules provides an additional data isolation mechanism. Thus, there are a total of two mechanisms—the object interfaces and the APIs—that isolate application processes from data management and communication services.

The **System Executive** provides high-level system control of SW processes. The complexity of this function can vary widely. In the case of simple functionality, it provides for the launching, killing, and status monitoring of processes. It may also

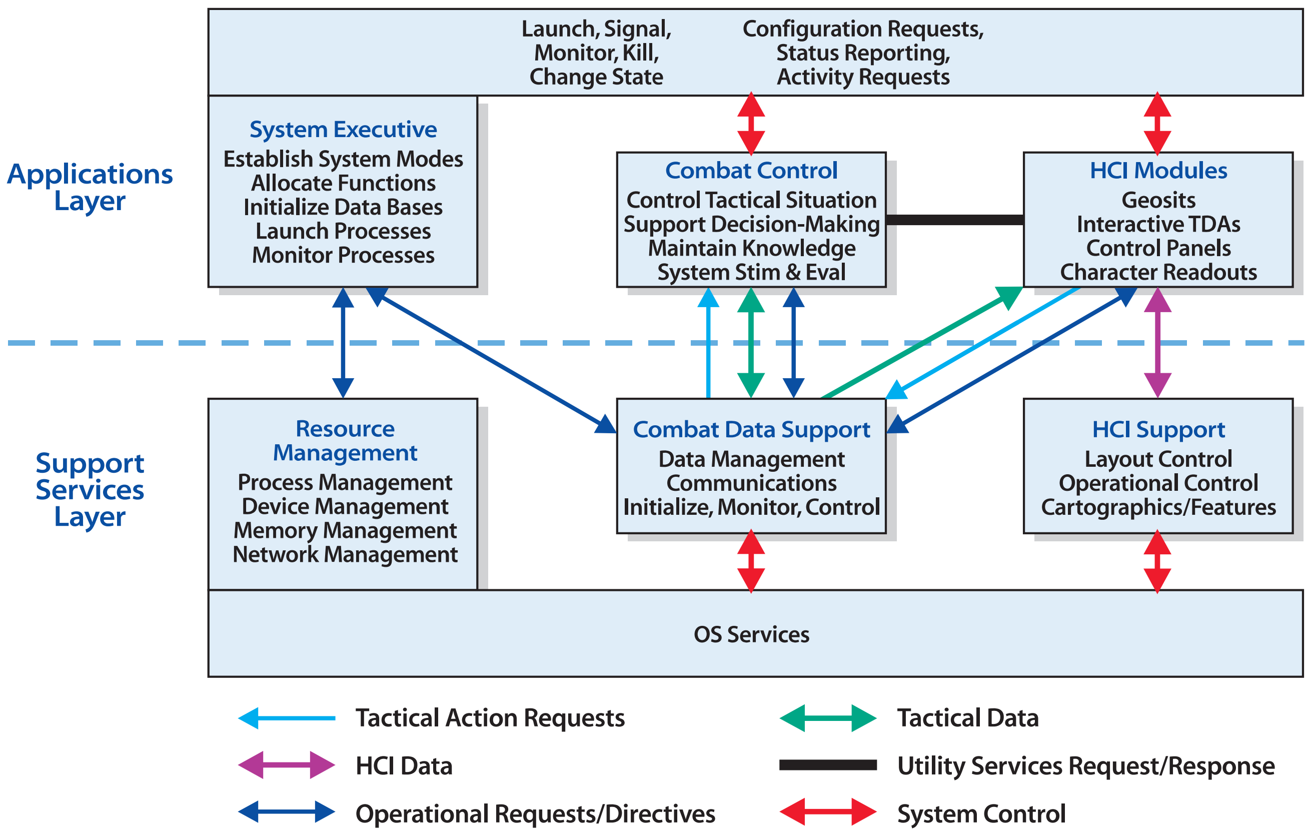

Figure 2—Combat Control Design Concept

provide for a system reconfiguration capability upon system initialization. However, there are also various types of dynamic system reconfiguration capabilities that require more complex functionality. One example of this is dynamic scaling of combat control processes according to operational conditions. Another example is the dynamic reallocation of combat control processes (to computers) according to system fault conditions.

As the complexity of the **System Executive** increases, so does the degree of coupling of this function with the combat control and display applications. This is due to the increasing amount of control and status information that must be communicated. In addition, the applications themselves may require embedded functionality to support process scaling and reallocation capabilities.

It cannot be assumed that the **Combat Data Support** system should have capabilities and performance attributes for handling system control and status information (unless by definition). That is, the requirements to support the transport and storage (e.g., triggers) of control data may be very different than the requirements to support similar handling of combat data. Because of their tight coupling, these alternate data-support mechanisms are characterized in Figure 2 as extensions to the **System Executive** itself. Design of **System Executive** type functions requires a high degree of program design expertise.

Figure 2 identifies the control functions of Figure 1 as part of the **Combat Control** applications. **Stim & Eval** (Stimulation and Evaluation) functions have been added to support system test, maintenance, and training modes. **Combat Control** applications are supported by (global) communications and data management services in the *Support Services Layer*.

There are instances where tactical data is to be displayed but need not be (temporarily) stored in the data management system (*Utility Services Requests* in Figure 2). An example of this are tactical predictions in support of running "what if" scenarios. The database schema cannot possibly support all possible variations of this. However, such direct interprocess communications between dissimilar functions should occur through a distinct and finite set of utility modules.

In the object paradigm, **HCI Modules** are typically display objects that correspond to the tactical objects (tactical picture, assessments, and plans) produced by **Combat Control** application objects. The (data) objects stored in the global data management system thus can represent the saved state of both the HCI and **Combat Control** objects. The desirability of defining composite objects for tactical applications and display is an area for research.

Each of the types of **HCI Modules** listed in Figure 2 can be designed as separate modules and implemented as object classes. It is typically desirable for combat systems to have reconfigurable consoles and displays. This capability can allow for "transparent control" of combat system functions from any station.

Support Services Layer

The *Support Services Layer* provides for OS services and the combat system "middleware." As shown in Figure 2, these middleware services are partitioned into **Resource Management**, **Combat Data Support**, and **HCI Support** functions.

There are many ongoing efforts in the Navy aimed at utilizing COTS OS and middleware services for ship systems. Use of COTS in combat systems will generally occur in a bottoms-up fashion. That is, it will first occur for computers and networking services, then OS services and, finally, middleware services. In addition, "cotsification" will first occur for those ship functions that are the least mission critical and have the least stringent requirements for real-time and fault tolerance.

Appropriate criteria for "make versus buy" decisions need to be established for all system components. The key in establishing these criteria is an understanding of the limitations of commercial equipment and SW for military use, and the possible impacts of these limitations on total system performance. In addition, module reuse of any kind has unique types of risk that must be identified, tracked, and mitigated.

Resource Management functions are differentiated in Figure 2 from **System Executive** functions by their direct interface with either devices and/or OS services. In this respect, **Resource Management** functions can add significantly to system complexity. The direct interfaces of **Resource Management** functions to devices and the system OS makes its development, maintenance, and testing particularly difficult. **Resource Management** may include monitoring and control functions for the purpose of increasing a system's fault tolerance characteristics. However, to be effective, these functions must have greater reliability than the combat system functions that they manage.

The **Combat Data Support** functions of Figure 2 provide the system communications and data management services. To the extent possible, these functions should have as much built-in fault tolerance capability as possible in order to assure uninterrupted availability of combat system data. For communications, this may involve automated link and node failure detection and rerouting.

This also applies to the underlying communications or data distribution capabilities of combat data management system. For data management functions, requirements for fault tolerance may impose additional requirements. As an example, the data management system may require automated capabilities to reestablish database consistency (of replicated data stores) after link repair or rerouting.

An important architecture issue is that of using (distributed) data management versus communications for combat systems integration. Use of either databases or high-level communications protocols promotes the structuring, definition, and thus comprehension of ship's data. However, databases have the important additional capability of maintaining relationships between data (or data objects).

Systems that use global data management for data distribution have the potential to be much more integrated than those that rely heavily on interprocess communications. This is because it is difficult to establish and maintain relationships between data produced by processes or subsystems without a global (common) database. Federated systems can be characterized as having subsystems that are interconnected utilizing interprocess communications alone.

Systemwide global data management imposes a systems discipline on developers that both promotes and supports integrated development. Global data management inherently breaks down "stovepipes" by discouraging data ownership. This is important in achieving the level of integration required for global ship functions such as resource management or multiwarfare common (consistent) tactical picture formulation. A globally available pool of data also provides incentives for developing new functionality.

Combat system design is, in many ways, equivalent to organizational design. Indeed, the combat system HW and SW must reflect ship command and operational structure. There are two fundamental types of organizational structures: functional and divisional. The divisional-type organizational structure is also referred to as product-oriented structure. In the division structure, the individual divisions are fairly self-contained and incorporate all the production-type functions needed for product development. In the functional organization, the functional units are pooled to produce the various products of the organization.

Ship systems today are generally organized along warfare areas, which corresponds to a divisional-type organization. This is often referred to as "stovepiping." The primary advantage to this structure is that it requires much less communication and coordination than the functional structure. However, there are also disadvantages. The divisional structure can have much functional redundancy because of the need to be self-contained. The other major disadvantage is that it is less flexible, requiring reorganizations as product-lines change. As it relates to combat systems, the divisional structure is less flexible in supporting new warfare areas.

With the new data management and communications technologies available today, there is less rationale for warfare area divisions. Ship control centers can be supplemented with these integrating data support systems. This supports a gradual migration towards an integrated combat capability. Combat system integration can provide capabilities to support new warfare areas based primarily on existing functionality.

It should be recognized that developing a communications and data management infrastructure to support future missions and warfare capabilities has certain risks associated with it. This is because the performance requirements of such an infrastructure are typically derived from mission requirements. In addition, rapid advances in technology can quickly cause existing systems to become obsolete. However, these risks are an inherent part of the development of any data-support infrastructure. Indeed, the telecommunications industry must continuously deal with this type of technology risk.

Figure 2 also identifies **HCI Support** as part of the combat system *Support Services Layer*. **HCI Support** provides functional-type services for the HCI modules. **HCI Support** functional modules should also be architecturally layered in order to simplify design. For portable operating system interface (POSIX)-compliant systems, the lowest layers would typically be X-Windows and MOTIF. On top of this layer would be display formatting or similar display control functions. These functions include:

- ❖ Control functions for HCI layout, menus, and variable action buttons
- ❖ Operational support functions for handling alerts and status information, and providing help services
- ❖ Drawing functions for environmental features/ maps and control panels

A challenge for the Navy is to begin developing standard libraries of these **HCI Support** modules. Each module should be kept simple (small), be well tested and documented, and maintained under configuration control. These modules could be used for the development of both ship systems and Advanced Development Models (ADMs). This would reduce much of the costs associated with prototype development and ADM transitions, and aid in new systems integration at the operational level.

Operating System Services

The *Support Services Layer* in Figure 2 is supported by OS services. These services include the computing platform OS and various OS extensions, such as file management services. Combat systems have special requirements for real-time, fault tolerant, and in some cases, secure computing. Each architectural layer must have capabilities in these areas in order to support the next higher layer. In those cases where the machine OS does not directly support certain aspects of system performance, it can impose the need for complex functionality for both the *Support Services* and *Applications* layers.

It has been theorized that greater overall system performance-to-cost could be achieved if the various performance aspects of a combat system could be made flexible. That is, if the various performance aspects of the system (real-time, fault tolerance, and security) could be balanced in (near) real time according to operational conditions and/or damage-conditions. This has been termed Quality-of-Service (QoS). QoS supports the notion of adaptable system services capable of arbitrating competing requests for system resources. However, achieving this advanced system concept depends heavily on the capabilities of the OS.

To ensure that future commercial Os's incorporate capabilities that are required by military applications, the Navy must adopt an "open" OS interface standard such as POSIX. The minimal requirement for "openness" is that any interested organizations can participate in the development of the (interface) standard. In addition, the Navy should participate in these standards committees as necessary to help ensure that they will incorporate the unique capabilities required for combat systems. Participation in standards committees requires very little resources and supports the increasing use of COTS in military systems.

ARCHITECTURE IMPLEMENTATION ELEMENTS

For developmental items, implementation is defined here as meaning detailed design. In the case of nondevelopmental items, implementation refers to the allocation of defined components to design elements. It might be difficult to imagine cases where it would be appropriate to have implementation details incorporated into the combat system architecture. Indeed, the general characterization of architecture as a top-level design seems to discount this possibility.

However, in the C4I community, architecture is often defined in terms of actual implementation. For example, the Defense Information Infrastructure Common Operating Environment (DII-COE) specifies the use of developed "common modules" and a specific, networked computing design. Even application modules, such as the tactical decision aids that are part of Joint Maritime Command Information System (JMCIS), are specified by DII-COE to be used for certain functions.

Thus, it is not unprecedented to have implementation-specific requirements levied on military systems. In addition, there may be justifiable reasons for wanting to do so. Development efforts can be greatly reduced if only application modules need to be developed. For example, Microsoft Windows is a COE for PC applications. Referring to Figure 2, this environment roughly corresponds to all functions except those of **Combat Control** and **HCI Modules**. Thus, a COE can provide much more than just OS services.

However, the appropriateness of specifying use of particular standards, components, and technologies depends heavily on the requirements of the combat system. For the C4I systems of today, their main role is in providing communications and data management services. In addition, timing requirements are relatively lax. Thus, for C4I systems, implementation details may be appropriate for establishing system architecture.

In the case of AAW systems, real-time requirements for the applications impose real-time requirements on other system components. This is certainly the case for the computing infrastructure. However, real-time requirements also impact the design the HCI and the supporting graphics utilities. The HCI must provide real-time display of radar sweeps or other fast graphics displays. In addition, operator interactions with tactical consoles must be fast. This is in contrast to the analysis-oriented HCIs of C4I systems. For example, JMCIS incorporates pull-down menus and other HCI features that cover the situational displays that can require much time to open and close.

Mandating the use of particular design or implementation elements must be done only with careful consideration. However, providing information on component technologies and interface standards is always useful to the architects and designers of a combat system. It can be especially useful when additional information on standards is provided regarding their use in a particular domain (military systems). This kind of information is often referred to as "profiling" a standard. Although profiles might not be considered a system architecture element, they can be extremely useful for designing and implementing combat systems.

CONCLUSION

Combat system architecture definition is the precursor to design. It conveys system requirements for what the system must do that include functional, data, and control characteristics. Combat system architecture may have elements of functional analysis, conceptual design, or implementation. Both the requirements that are conveyed and the specific types of elements that are used to convey them depend very much on the requirements themselves.

In general, functional and process analysis models should be used to specify combat system requirements for functionality. Design elements of architecture identify specific functions and their organization. Models of architectural layers, functional partitionings, and (ultimately) object class structures are developed to convey combat system design requirements. In each case, interfaces between modules and/or partitionings are identified and defined. Combat system architecture incorporates design elements as needed to convey the required quality attributes of the system.

In some cases it may be desirable to include implementation details in the architecture definition. However, it should be recognized that this is effectively implementing parts of the system prior to design. Defining ship middleware that is capable of supporting multiple combat systems is one area where this may be justified. Middleware is the key to integrating existing combat system functionality in supporting future missions and warfare areas.

BIBLIOGRAPHY

Pollard, J.R., *Combat Systems Vision 2030: A Combat System Architecture for Future Surface Combatants,* Naval Surface Warfare Center, Dahlgren Division Technical Report TR 91-607, Dahlgren, VA, Sep 1991.

THE AUTHOR

MR. JAMES D. HORNER

Mr. James D. Horner received B.S.E.E. and M.S.E.E. degrees in computer engineering and an M.B.A. in information systems from the University of Maryland in May 1983, December 1986, and May 1991, respectively. From 1991 to 1994, he served as Deputy Director of ASW Technology Programs at the Naval Surface Warfare Center, Dahlgren Division, White Oak. Since then, Mr. Horner has been providing systems engineering expertise for the transitioning of combat system technologies and related development processes for various Navy technology programs. He is a member of the Fredericksburg Chapter of the International Council on Systems Engineering.